A STUDY OF IMPLEMENTATION METHODOLOGIES FOR DISTRIBUTED REAL TIME COLLABORATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Lauren Craft

June 2021

COMMITTEE MEMBERSHIP

TITLE: A Study of Implementation Methodologies for Distributed Real Time Collaboration

AUTHOR: Lauren Craft

DATE SUBMITTED: June 2021

COMMITTEE CHAIR: Maria Pantoja, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: John Clements, Ph.D.
Professor of Computer Science

ABSTRACT

A Study of Implementation Methodologies for Distributed Real Time Collaboration

Lauren Craft

Collaboration drives our world and is almost unavoidable in the programming industry. From higher education to the top technological companies, people are working together to drive discovery and innovation. Software engineers must work with their peers to accomplish goals daily in their workplace. When working with others there are a variety of tools to choose from such as Google Docs, Google Colab and Overleaf. Each of the aforementioned collaborative tools utilizes the Operational Transform (OT) technique in order to implement their real time collaboration functionality. Operational transform is the technique seen amongst most if not all major collaborative tools in our industry today. However, there is another way of implementing real time collaboration through a data structure called Conflict-free Replicated Data Type (CRDT) which has made claims of superiority over OT. Previous studies have taken place with the focus on comparing the theory behind OT and CRDT's, but as far as we know, there have not been studies which compare real time collaboration performance using an OT implementation versus a CRDT implementation in a popularly used product such as Google Docs or Overleaf.

Our work will focus on comparing OT and CRDT's real time collaborative performance in Overleaf, an academic authorship tool, which allows for easy collaboration on academic and professional papers. Overleaf's current published version implements real time collaboration using operational transform. This thesis will contribute an analysis of the current real time collaboration performance of operational transform in Overleaf, an implementation of CRDT's for real time collaboration in Overleaf and an analysis of the performance of real time collaboration through the CRDT imple-

mentation in Overleaf. This thesis describes the main advantages and disadvantages of OT vs CRDTs, as well as, to our knowledge, the first results of a non-theoretical attempt at implementing CRDTs for handling document edits in a collaborative environment which was originally operating using an OT implementation.

ACKNOWLEDGMENTS

Thanks to:

- My parents, family and friends for their constant love and support
- Maria Pantoja, for her advice, time and assistance as my advisor on this thesis
- Franz Kurfess and John Clements for taking the time to be on my committee
- Andrew Guenther, for uploading this template

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The following chapter provides an introduction into distributed real-time collaboration, its importance and the contributions and focus of this thesis.

1.1 Importance of Effective Tools for Collaboration

Collaboration is an essential part of everyday life. It serves as an important tool for many different areas of work including business, science, research and technology. It is the opportunity and success that comes from collaboration that helps to drive innovation in our society today. Collaborative editing allows tasks to be completed in a shorter period of time, the total number of errors to be reduced, and more view points and skills to be included in the end result [2, 6]. It is nearly impossible for a professional to avoid participating in collaborative work. Therefore in order for collaboration to be as successful and productive as possible, collaborative platforms must allow for real-time updates which are tolerant of faults and mistakes. A good collaborative system should allow a user to perform edits on a shared document as easily as they could perform edits on a single author document [2, 5].

Collaborative tools allow employees and/or researchers to work together on the same project no matter their physical location. The ability for humans to work together on a single task from different locations has always been important, but its importance has increased since 2020 when a global pandemic (COVID-19) hit the entire world. Within weeks of the first local cases being detected, most of the US work force had

to transition from working in their company office to working remotely from their homes.

An obvious part of collaborative tools, compilers, text editors, etc.; is that each of these software tools is a Distributed System (DS). A distributed System is difficult to define, but most DS designers agree that DS is a system with multiple different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user. Many of these DS present the same coordination and communication challenges; in this thesis we will specifically focus on problems related to distributed collaboration and the need to synchronize documents (code, text, etc) that will be written by multiple people. The distributed collaboration tools discussed in this thesis allow for multiple clients to collaborate on document replicas in a manner that emulates, as nearly as possible, the same effectiveness of collaboration as when one client is performing edits on a non-replicated document [3]. Distributed Collaboration requires infrastructure, such as emails, instant messaging and document sharing platforms. Since Document Sharing platforms are becoming an increasingly important type of collaborative environment and since they are a relatively "newer" technology, there has been a lot of recent research in this area. Collaborative tools present very interesting challenges from the Distributed Systems point of view. In this thesis we wanted to explore some of the more recently published algorithms meant to improve distributed collaboration so that we can compare and implement the algorithms by benchmarking their real performance on overleaf, a distributed collaboration tool used by millions of researchers around the world.

What is the main problem in regards to distributed collaboration tools? Most articles focus on project management problems, for example, how to keep teams motivated or how to effectively communicate with each other [30]. Those problems are interesting, but we want to focus on the technical aspects of distributed document creation.

The aforementioned technical aspects which cause problems are mainly attributed to synchronization and consistency. In order for consistency to be maintained amongst distributed copies of a document, there are three requirements which must be met: convergence (all documents result in the same state once edits are applied), intention preservation (the effects of edits are as the user intended) and causality preservation (all edits are applied in their causal order). There are many factors which can cause consistency to fail which will be discussed in this thesis. One example is if two users are editing a document in the same location but the network connection is bad and there is high latency, each user may think they are writing in that spot alone with no conflicts, but their edits may be becoming jumbled with each other since they are not able to see the effects of each others edit in close to real-time. Therefore, it is difficult to implement a platform in which a single document can be modified by different engineers/researchers at potentially the same time. These technical problems are described in further detail in the related works and discussion chapters.

The problems which come with implementing distributed collaboration are by no means trivial. Some of the most commonly used distributed tools such as Google Colab [9] or Jupyter Notebooks [15], do not allow multiple collaborators to work on the same project. Engineers at these companies have been working on a collaborative version of their respective platforms for several years now. For this thesis we wanted to initially study adding a collaborative capability to Google Colab, but Google no longer provides an open source version of Colab thus making it impossible to further develop with their platform. Overleaf [10] on the other hand, does provide the source code and already has collaborative capabilities. Since Overleaf is a commercially available tool being used by millions of researchers, its open source and collaborative aspects made Overleaf the perfect candidate for this thesis, However, since Overleaf was developed by "hundreds" of experienced engineers and since it is a large and complex piece of software, it took a long time to become familiar enough with Overleaf to the point

that we were able to begin performing changes to the communications/distribution of Overleaf.

1.2 Techniques for Real-Time Collaboration

There are various techniques to allow for real-time collaboration, but the two most common and effective types are through operational transforms (OT) [**?**] and conflict-free replicated data types (CRDTs) [21]. The operational transform technique has been around much longer than CRDTs. The first paper which introduced the OT technique was published in 1989 and was widely adopted in 2009 into main tools like Apache Wave and Google Docs [7]. As for CRDTs, the first article which introduced the idea was published in 2011. The idea of a CRDT is slowly being adopted and researched by various distributed companies, but many of the companies are yet to fully adopt CRDTs for use in implementing real-time collaboration [21].

Most major collaborative platforms implement the operational transform technique [11, 2, 16]. When CRDTs were proposed years after the operational transform methodology was proposed, there were claims that CRDT solutions were better suited for real-time collaboration than OT solutions. Some even went so far as to say using OT for distributed collaboration was a wrong and inefficient technique [25, 27, 28]. Despite CRDT claims of superiority over OT, there are no major products which have disposed of their OT collaborative implementations and migrated to usage of CRDTs.

1.3 Our contributions

The work outlined in this thesis is aimed to understand why CRDTs are not commonly used for real-time collaboration implementations in major commercial products. We focused on Overleaf, an academic authorship tool, which allows for real-time collaboration using operational transform. Our contributions include an evaluation of Overleaf's operational transform implementation in respect to real-time collaboration, a CRDT implementation of real-time collaboration integrated into Overleaf and an evaluation of how the CRDT implementation performed in comparison to the OT implementation. We wanted to evaluate if CRDT-driven real-time collaboration performed better, worse or the same as operational transform driven real-time collaboration. The results of our research and experiment are outlined throughout this paper. We also provide a discussion of the respective advantages and disadvantages of OT and CRDTs.

1.4 Outline of Chapters

Following this introduction is a background section which provides an overview of the important topics and technologies which will be addressed in this thesis. The next chapter after the background is the related works section. This section will discuss previous work performed by researchers into the theory and performance of operational transform and conflict-free replicated data types. Following the related work is where our contributions begin. Chapter 4 discusses the implementations of OT and CRDTs in Overleaf. Chapter 5 then evaluates the performance of the OT and CRDT implementations of real-time collaboration in Overleaf. Following the evaluation, Chapter 6 is where discussion takes place in order to compare and contrast the two ways of implementing real-time collaboration. Chapter 7 is where

conclusions will be drawn and discussed. The final chapter, Chapter 8, acknowledges possible areas of expansion and ideas for future work and further research to expand upon the work performed in this thesis.

Chapter 2

BACKGROUND

This chapter describes in detail some topics and pieces of information which are necessary for proper understanding and agreement when reading this thesis.

2.1 real-time Collaborative Tools

Real-time collaborative editors allow multiple users to simultaneously edit documents no matter their respective geographic locations [27]. Modern collaborative editors have risen in popularity over the past decade as a necessary tool for students and professionals alike. Collaborative tools provide the ability for teams to get work done more efficiently with reduced error and an increase in the variety of viewpoints and skills which contribute to the work [6, 2]. The need for collaborative tools is especially high during the transition to remote work during the COVID-19 pandemic. However, despite the more recent need and motivation for collaborative editors, the idea of a collaborative real-time editor has been around for some time now. The first real-time collaborative editing tool was proposed by Douglas Engelbart in 1968 in the 'Mother of All Demos' [7, 20]. Engelbart's 'Mother of All Demos' was presented at the Association for Computing Machinery / Institute of Electrical and Electronics Engineers (ACM/IEEE) Computer Society's Fall Joint Computer Conference in San Francisco. At this conference, the idea of collaborative editors was applauded, yet it was many years after when a collaborative editor was actually created. In order for collaborative editors to exist, techniques and structures to handle consistency and errors needed to be developed and tested. Without a strong technique to handle these situations

and ensure that each client is receiving the correct updates, real-time collaboration would not be effective. Another essential aspect for effective distributed real-time collaboration is the edit latency and performance. If the collaborative platform does not efficiently respond to edits in a manner which is comfortable to the user, users will become frustrated and their work will not be effective [6, 2, 13]. The consistency and performance of distributed collaborative systems will be discussed throughout this thesis.

2.2 Operational Transforms

Operational transform (OT) is the replication mechanism which real-time collaborative applications traditionally rely on [2]. The operational transform technique was made by C. Ellis and S. Gibbs in 1989 for their Group Outline Viewing Editor (GROVE) [6]. Ellis and Gibbs intended for the operational transform technique to allow for maintenance of consistency and concurrency in collaborative editing environments. Collaboration with operational transforms relies on replicated document storage. This means that each client has their own copy of the document which operates locally with no locking or blocking mechanisms. The changes executed on the client's document are then propagated to the other clients. Each co-editing site maintains a local internal buffer. The buffer begins empty, but once a user makes a change to the document, the local operation handling begins and the operation is placed in the client's local internal buffer. Once a change is added to the client's local internal buffer, the change must be transformed in accordance to concurrent operations performed by other replicas in order to ensure intended operation is preserved and to allow for proper convergence on all replicated document sites. Once the operation has undergone the transformation step, it is then propagated to the other clients to be added into their local buffers and incorporated into their local copies.

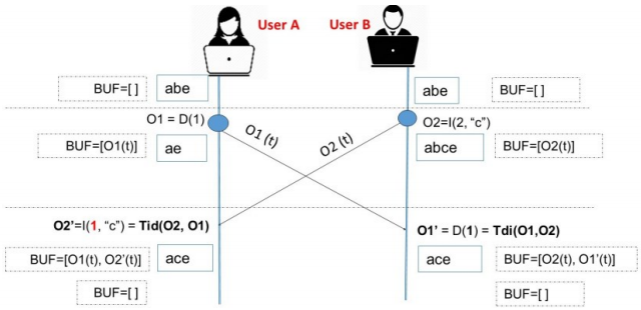


Figure 2.1: This figure demonstrates the basic idea of how the operational transform algorithm achieves the general transformation of a given edit. Original image from [25]

Figure 2.1 above demonstrates a simple example of the text "abe" being manipulated by two clients in a collaborative text editor. Each of their external states begin with "abe" and their local internal buffers begin empty. We assume that position indices begin at 0. In this example, client 1 deletes the character 'b' from position 1, while client 2 simultaneously inserts the character 'c' in position 2 in the "abe" text. Assume that the two operations are concurrent and take place without any knowledge of each other [25]. When client 1 deletes character 'b' from position 1, a timestamp is created for that operation. At the same time as client 1's deletion operation is executed, client 2 is performing its concurrent insertion operation, inserting the character 'c' into position 2. OT has an internal ordering algorithm which determines what order to handle concurrent operations. We will assume the algorithm decides to handle client 1's update first. So client 1's operation will be applied to create the resulting string "ae". Next client 2's operation must be transformed in accordance to how the concurrent operation performed by client 1 affects it. Since client 1 deleted a character at index 1 and client 2 wants to insert a character at a later index, client 2's index must be shifted by 1. The resulting operation after the transformation is to now insert 'c' into position 1. Once this operation is applied to the text the resulting text is "ace". The operations are all position-based in relation to the positions of the document's external state. Therefore, when operations are concurrent, the local

internal operation positions must be transformed so that they agree with the users' intention and allow for consistency amongst all clients. Once the transformation functions take place, the respective client's operation is placed into the client's internal buffer. The system then propagates the operations out to the other clients. Once both operations are propagated to the clients, their buffers are updated accordingly. This allows for the clients to each undergo operations which have been coordinated with the operations performed by all clients editing the document. In the event that two users concurrently perform an operation in the same location (i.e. client 1: "abe" becomes "ace" and client 2: "abe" becomes "ade"), most OT implementations have an internal ordering algorithm to decide the order in which to handle and apply concurrent edits. Another possible option is to have an algorithm which decides which edit wins and then do not apply the edit which loses. However, this results in data loss which may not be favorable. No matter how the OT algorithm decides to handle concurrent edits, the OT technique allows for consensus and correctness during collaboration in the text editor.

2.3 Conflict-free Replicated Data Types

Conflict-free replicated data types (CRDT), sometimes also referred to as Commutative Replicated Data Types, are a more recently introduced class of replication mechanisms. CRDT operations execute concurrently and are specifically designed to be commutative, meaning they rely less heavily on consensus and synchronization [2]. The concept of CRDTs was first formally defined in 2011, years after the concept of OT was first defined. The concept of CRDTs was created with the intention of it allowing for a more efficient and correct technique for collaborative text editing. All approaches to CRDTs are supposed to provide strong eventual consistency. Eventual consistency is the idea that each client or user that is editing their own local copy

of the collaborative environment, which allows each client or user to perform local, tentative updates. These updates are periodically propagated to the other clients at times that the connections are available [4]. Strong eventual consistency is a consistency model which guarantees that all operations and edits are propagated to the other clients in the correct sequential order. This approach to collaborative changes allows all user edits to eventually be caught up and available to each client or user.

The general idea behind how CRDTs operate and handle updates is as follows. A user or client performs an edit on the document. This edit is registered as a local operation and the effective change of the edit appears immediately on the user's document. The operation which was generated is first in a position-based operation, it is then transformed into an identifier-based operation to then be added to the internal data object sequence representing the current document state of that user or client. The identifier-based operation which is created is then propagated to all remote clients. When a remote client receives the identifier-based operation, the operation is applied to the remote client's internal object sequence, the operation is converted from identifier-based to position-based, and given the operation, the corresponding edit is applied to the remote client's document state.

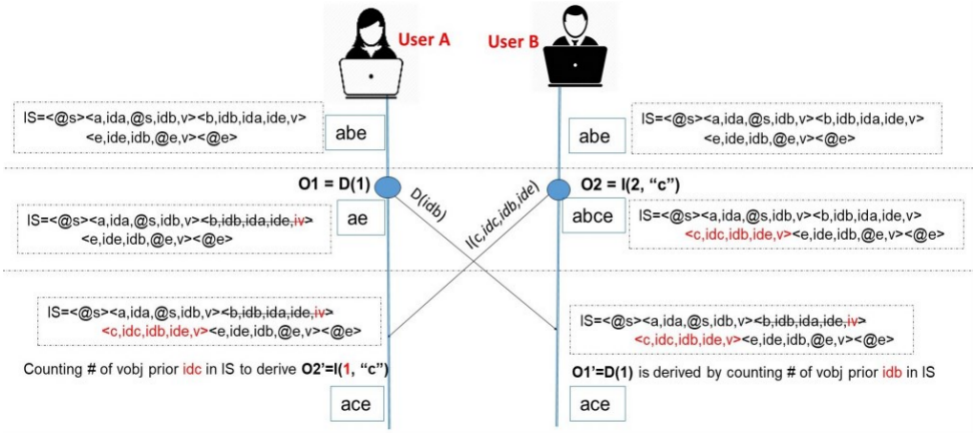


Figure 2.2: This figure demonstrates the basic idea of how the conflict-free replicated data type algorithm achieves the general transformation of a given edit. The original image is from [25]

Figure 2.2 above demonstrates an example of how a CRDT handles concurrent operations performed by two clients working on a text editor which begins with an initial state of the string "abe". For CRDTs, the initial data object sequence is populated according to the initial state of the document. In the context of this example, the data object sequence begins as:

IS=[@s],[a,ida,@s,idb,v],[b,idb,ida,ide,v],[e,ide,idb,@e,v],[@e]

For each object in the data object sequence, there are 5 distinct and required parts: if we look at ¡b,idb,ida,ide,v¿, b is the character which the object represents in the document, idb is the immutable identifier assigned to that object, ida and ide are the immutable identifiers of the objects which immediately surround the idb object and lastly, v is what indicates that the object is visible in the document. If the object were to be deleted from the document at a later time, the v would instead become iv for invisible and the data object becomes known as a tombstone since it is no longer live and viewable in the document. As for @s and @e objects, those are special identifier objects which mark the start and end of the internal object state.

As edits are performed by the clients, the internal data object sequence is updated accordingly. We assume that the position indices begin at 0. For this example, client 1 performs an operation to delete the character at index 1, this results in the string "ae" since the b at index 1 was deleted. Concurrently, client 2 performs the operation to insert the character 'c' into index position 2 which generates the resulting string "abce". Each of their position-based operations are converted into identifier-based operations. This conversion to identifier-based operations is completely internal and not visible to either client. Once the identifier-based operations are generated, they can be incorporated to the internal data object sequence. Client 1's operation is first applied to the internal object sequence. For this operation, the identifier-based

operation becomes D(idb) indicating the object with the identifier idb must be turned invisible. This results in the internal object sequence becoming:

IS=[@s],[a,ida,@s,idb,v],[b,idb,ida,ide,iv],[e,ide,idb,@e,v],[@e]

The identifier-based operation of client 1 is then propagated to all other clients. Once a client receives the propagated identifier-based operation, it is converted to a position-based operation and applied. Then client 2's identifier-based operation, I(c,idc,idb,ide), is applied. This identifier-based operation was created by examining the internal object sequence and identifying the neighboring object identifiers to index 2 where the object is to be inserted, idc is the resulting immutable identifier for the new object to insert. To insert the new object, the number of visible objects in the internal object sequence which were present before the current operation and all operations concurrent with this operation are counted until we reach the correct position. The new object ¡c,idc,idb,ide,v¿ is added to the object sequence. This results in the following object sequence:

IS=[@s],[a,ida,@s,idb,v],[b,idb,ida,ide,iv],[c,idc,idb,ide,v],[e,ide,idb,@e,v],[@e¿]

The identifier-based operation is then propagated to all clients, converted to a position-based operation and applied just as client 1's operation was. The communication structures which operate with CRDTs are not specifically designed, this leaves more implementation and design decisions for the programmers. Overall, the CRDT algorithm is an identifier-based internal data object sequence style which continuously grows with the document. The space and time complexities of CRDT algorithms will be discussed in Chapter 6.

2.4 ShareJS

ShareJS is an open source operational transform library which can be utilized to add real-time concurrent editing to web applications [8]. ShareJS handles all of the edits which are submitted and applies them to its internal master document which is contained inside a ShareJS database. This specific operational transform library, ShareJS, is what Overleaf uses for its collaborative editing functionality.

2.5 Automerge

Automerge is an open source conflict-free replicated data type library for adding real-time concurrent editing to applications [14]. Automerge is still under development and is continuing to make advancements in its CRDT implementation. This library is what our thesis relies on for implementing real-time collaboration in Overleaf through a CRDT methodology.

2.6 ShareLaTex

ShareLaTex is a collaborative web LaTex editor which was acquired and integrated into Overleaf in 2017 [29] Before integrating ShareLaTex into Overleaf, the two products had similar purposes and goals of acting as a real-time collaborative LaTex editing environment. From our research we believe that Overleaf implements its collaborative features and document updates using ShareLaTex source code. In order for Overleaf to maintain its documents, there is a connection to a Redis ShareLatex database. Redis will be defined and described in detail below.

2.7 Redis

Redis is a data structures server which is used by ShareLaTex to store the LaTex document source [17]. The Redis server is responsible for storing the document versions which are displayed on the front end for Overleaf. Each client has a view of the document at its current state in Redis. Each time a change is made, the document is retrieved from Redis, updated by the document-updater container, and then sent back to Redis with its new changes applied.

Chapter 3

RELATED WORK

The following chapter describes previous work done by researchers to look into applications and comparisons of operational transform and conflict-free replicated data types.

3.1 Comparisons of OTs and CRDTs

Previous research has been performed in attempt to form comparisons between the two replication mechanisms: operational transform and conflict-free replicated data types. Much of this research focuses on evaluating the more recently developed CRDT algorithms since less is known of its performance and usability in comparison to the current knowledge of the traditional OT algorithms. As far as we know, from our research it appears that the previous work in the area of comparing OT and CRDT performance for distributed collaboration has been largely theoretical. We have not found examples where researchers have taken an operational transform implementation and re-implemented the collaboration mechanism to utilize a conflict-free replicated data type, and then compared and evaluated the two implementation methodologies in respect to each other.

Prior research has taken place in attempt to examine the real differences between the long used and highly regarded operational transform algorithm, and the newer and more obscure conflict-free replicated data type method for working with distributed collaborative editors. There have been claims of purported advantages which CRDTs maintain over OT in the past which have caused a lot of confusion and have left

room for debate upon the validity of the statements of superiority in the past [27]. A team comprised of researchers from Nanyang Technological University in Singapore, Singapore and Codox Inc. in San Jose, CA, USA, worked to reveal the differences between the CRDT and OT approaches and uncover the validity of various claims and myths in regards to the performance of CRDTs in collaborative editors. Their work was published in a three-paper series of articles [28, 27, 25].

In the first article of the series, the researchers investigated the differences between OT and CRDT approaches under a general transformation framework for consistency maintenance in collaborative editors which they developed by examining the core techniques and basis for multiple OT and CRDT techniques [25]. In today's modern world, real-time distributed collaborative editors all maintain similar underlying architectures with an editor application and shared documents which are replicated and distributed to each of the collaborating clients editing sites [25]. When a client is working on a collaborative document, their edits typically appear instantaneously upon their local version, then the local edits are propagated to the remote sites in as close to real-time as possible. The edits which are propagated to the other editing sites can either be distributed as 'operations' or 'states'. For the OT and CRDT algorithms, the edits are treated as operations. Therefore, the main problem to consistency is to find a way for the operations generated by one local document replica, to be transformed and replayed on all other replicas while maintaining consistency across all replicas. The details in how the OT and CRDT algorithms handle their editing operations in order to maintain consistency amongst all document replicas will be discussed below.

In their research, the team at Nanyang Technological University and Codox Inc found that CRDT is similar to OT in that they both follow a general transformation approach, but CRDT largely differs from OT in that CRDT indirectly achieves the

transformation, whereas OT directly achieves the transformation. This transformation approach treats each edit as an operation which must be transformed in some way into a newer version which will maintain consistency amongst all document replicas even during concurrent edits on replicated sites. The general transformation approach allows for concurrent operations to maintain a commutative property, meaning that they can be transformed in different orders, yet still yield the same results.

The general transformation approach which the researchers discovered is the generic sequence of steps which both OT and CRDTs follow can be described as follows. Consider a situation in which two users are working on the same plain-text document which is replicated at each of their individual editing sites. For this example, each individual editing site for the respective document maintains the text "abc". For the first example edit to the general transform methodology, user 1 performs the operation to delete the character at index 1. Then, user 2 performs the operation to insert the character 'f' into index 2. Assume that the two operations are executed concurrently. Each of the individuals operations appear instantly on their local version allowing for an immediate local response. However, the operations each client performed need to be transformed in such a way that they maintain consistency on all replicas which propagated to the various editing sites. User 1's operation is transformed according to how user 2's operation may affect it. Since user 2 is editing at index 2 and user 1 is editing at index 1, user 1's operation is not affected and does not need to be changed during its transformation step and can then be propagated to the editing site of user 2. The operation user 2 made needs to be transformed according to the edit which was made by user 1. User 2 wanted to insert a character at index 2, but user 1 deleted a character at index 1. Therefore, the indices of the content from index 1 on have all shifted left by one. The operation coming from user 2 to user 1 will be transformed into an insertion of the character 'f' at index 1. The transformation and propagation of the operations under this general transformation maintains consistency on all replicated

editing sites. The requirements for this consistency include convergence, meaning all document replicas are identical, as well as preserved intention, meaning the initial effect of the local operation is the same as the operations resulting effect once the operation is propagated and synced to the other editing sites.

The researchers at Nanyang Technological University and Codox Inc. made many discoveries about OT and CRDT which are relevant to this thesis. First they found that OT and CRDT require the same general consistency requirements of convergence, intention-preservation and causality-preservation. The meanings of these requirements and their importance will be discussed more in a later section. A difference they emphasized between OT and CRDT is the way in which their concurrent operation transformations are handled. OT implementations operate using a compare-calculate method [25] in which the numerical positions in their operations are compared and transformed to their new positional arguments based on how the concurrent operations affect each other. CRDT solutions are identifier-based and applied to an internal object sequence. The internal object sequence maintains a list of all visible objects, and some CRDT implementations also record the non-visible objects which have been deleted. For the context of this thesis we observe CRDT implementations which include both the visible and invisible data objects in their internal sequence. The operations performed by clients are transformed by the CRDT search-count method which scans the objects in the sequence and counts the number of visible objects and inserts or removes in accordance to the given operation. The researchers found that the OT compare-calculate method is more efficient than the CRDT search-count method in that the OT method has the same constant, low cost, whereas the CRDT method can be variable in length and maintains a high cost [25]. The aforementioned difference between OTs direct transformation approach and CRDTs indirect transformation approach can be explained by OT recording the impact of concurrent operations in the buffer and directly comparing and calculating the effects of all op-

erations based on their positional differences, whereas CRDTs record the effects of all operations in the internal object sequence, then convert the positional-based operation into an identifier-based operation for the local site application and then take the identifier-based operation, convert it back to a position-based operation and apply it to the remote site. Both theoretically produce the same result, but their approaches are very different. A table describing the similarities and differences between OT and CRDT for performing the general transformation for edits in collaborative editors is shown in Table 3.1.

Table 3.1: A comparison of the techniques and steps for OT and CRDT implementations performing the general transformation. The table is originally from [27].

	OT	CRDT
Consistency Requirements	Convergence and intention-preservation	
External State (ES)	A sequence of characters: *c0, c1, c2, ..., cn*	
External Operation (EO)	A pair of position-based operations: *Insert(p, c)* or *Delete(p)*	
General Transform (GT)	GT: $EO_{local} \rightarrow EO_{remote}$	
Commutativity	Concurrent EOs are made commutative by GT, not natively	
Internal Operation (IO)	Position-based operations by timestamping EO	Immutable identifier-based operations from converting EO
Internal State (IS)	A buffer of concurrent IOs, independent of ES	A sequence of objects (+ optional tombstones), mirroring chars in ES
GT Realization	Direct transformation: position → position	Indirect transformation: position → identifier → position

In the second article of their three article series, the researchers at Nanyang Technological University and Codox Inc. looked into the correctness and complexity of OT and CRDT. In their article, they first address their findings on the correctness of OT algorithms. In order for convergence of collaborative documents to be properly reached, there are two properties which must be met: Convergence Property 1 (CP1) and Convergence Property 2 (CP2).

- Convergence Property 1: Given Oa and Ob defined on the document state DS, and a transformation function T, if Oa'= T(Oa, Ob), and Ob' = T (Ob,

Oa), then applying Oa and Ob' in sequence on DS produces the same state as applying Ob and Oa' in sequence on DS. [27]

- Convergence Property 2: Given Oa, Ob and Oc defined on the same state, and a transformation function T, if Oc' = T(Oc, Ob) and Ob' = T(Ob, Oc), then transforming Oa against Ob and then Oc' produces the same operation as transforming Oa against Oc and then Ob'. [27]

Convergence Property 1 is easily achievable in a plain text setting, however, Convergence Property 2 can be more challenging to maintain depending on the editing circumstances. The challenge with maintaining convergence property 2 will here on be referred to as the CP2 issue. The specific case in the CP2 issue which can be non-trivial to handle is the false-tie puzzle. The false-tie puzzle was discovered and detailed by [26, 24]. The false-tie puzzle is one in which arbitrary transformation paths amongst multiple transformations result in different outcomes. This issue is solved by completely avoiding the requirement for CP2 at all by writing OT algorithms which do not arbitrarily transform concurrent operations in random orders, but rather have a system in place for the same order to apply transformations to concurrent operations in various contexts [27]. There is another approach to achieving and persisting CP2 through CP2-preserving algorithms which ensure the order of the transformations continue to preserve the convergence properties; these algorithms are much more complicated than the ordering approach. Despite claims of OT being incorrect and having challenges, the researchers found that over the years all problems and challenges to OT correctness have well-developed and tested solutions.

As for CRDT approaches, there are challenges in creating unique object identifiers. The CRDT approach requires each piece of content in the editor to be an object identifier with an identifier for the text character itself, as well as identifiers for its immediate neighbors. Adding in characters in the middle of two existing characters

in a document can result in difficulties with creating unique identifiers, especially if the identifiers are largely number-based. Since the numbers have limitations on precision, there are only so many numbers when choosing randomly between two existing numbers. This limitation can cause problems such as no available identifiers left to be chosen, as well as possible duplicate random number identifiers. Inconsistent and violating positions and identifier problems in the CRDT approach are still in need of solution development in order to ensure correctness [27]. This finding contributes to the research in this thesis in that it may be an indicator of problems which occur when a CRDT approach is integrated in Overleaf. The Automerge CRDT library used in this thesis claims to have solved the problem and found a solution for creating unique identifiers [13]. This solution will be discussed in detail in Chapter 7. There are other challenges with a CRDT approach which will be addressed when comparing OT and CRDT space and time complexities.

3.1.1 Analysis of Logs from Collaborative Sessions

Evaluating the performance of the operational transform algorithm in comparison to the commutative replicated data type performance has proven to be a useful area of study in order to gain a better understanding of how the two ways of implementing real-time collaboration compare. The ability to have an effective and easy to use collaborative editor is essential for team work in a variety of settings including academia, professional and personal. A good collaborative editor should be as easy to edit and work on as a single-user editor. Investigating further into various ways of implementing real-time collaboration can assist in reducing task completion time as well as errors [2].

A team of researchers from University of Lorraine in France focused their research on evaluating the suitability of CRDTs for collaborative editing. The researchers

first analyzed the theoretical performance of the CRDT algorithms in comparison to the OT algorithm. Much of the research that has been conducted on comparison of CRDT and OT algorithms has focused on the theoretical side. In order to form more concrete comparisons between various CRDT algorithms and the more traditional OT algorithm, the researchers conducted their experiment by evaluating the editing logs created from peer-to-peer collaboration environments for each of the algorithms that were being assessed. The researchers manipulated a text editor to output editing logs which recorded the sequence of edits performed by their experimental users.

Their experiment consisted of students simultaneously working on documents using a collaborative editor, during which many of the operations generated were logged as data to be evaluated upon experiment completion. In this experiment, TeamEdit was used as the real-time collaborative editor where the students performed their collaborations. The operations that were evaluated were narrowed down to character insertions and character deletions. Their work was the first experiment to analyze CRDT and OT algorithms using collaboration traces [2]. Once the collaboration traces were created by the students participating in the experiment, the various CRDT and OT algorithms which they were analyzing were then applied to the outputted collaboration traces.

The researchers essentially simulated how an implemented version of each algorithm would perform on the provided collaboration trace in order to compare the algorithm performances. By simulating their OT and CRDT algorithms on the collaboration traces, they were able to conclude that both OT and CRDTs appear to be suitable for real-time collaboration. Their research went beyond the more popularly seen theoretical evaluation of CRDT and OT performance, and applied the algorithms to the collaboration traces. However, they did not thoroughly explore the difficulties of concurrent operations and convergence issues. Their work mentioned future paths

of research to explore communication complexity and convergence latency [2]; areas which we look into in this thesis. The research discussed in this thesis will go beyond theoretical evaluation and explore actual implementations of an operational transform algorithm as well as a Commutative Replicated Data Type algorithm in the rich-text editor Overleaf.

3.2 Assessment of CRDT Performance

When CRDTs were first proposed in 2006, there were claims that CRDT solutions were more favorable than OT solutions. OT solutions began to be regarded as 'incorrect and inefficient technique' [28]. Despite the superiority that CRDT algorithms were said to hold over OT algorithms, CRDT algorithms are not popularly seen in practice amongst the popular collaborative editing tools created by the leading technology companies. The scarcity of CRDT algorithms used in real-world applications has led to confusion and a lack of consensus as to the true effectiveness of CRDT algorithms.

Chapter 4

IMPLEMENTING REAL-TIME COLLABORATION IN OVERLEAF

This chapter outlines the operational transform algorithm which was already implemented in the published version of Overleaf, as well as the CRDT algorithm which we implemented and integrated into Overleaf on our own in order to examine how the two ways of implementing real-time collaboration compare to one another.

4.1 Operational Transform

Overleaf, like most popular real-time collaborative document editing products, utilized operational transform (OT) for its method of implementing real-time collaborative features. Operational transform was developed as a method of implementing real-time collaboration in the late 1980's [25, 7]. OT involves distributed computing theories and concepts which specifically focus on concurrency and context [25]. Major commercial products for collaborative editing all utilize OT in order to implement real-time collaboration. Operational transform is the common way of implementing document collaboration despite claims that CRDTs may be better performing [16, 2, 11]. Previous work has examined the theory behind OT and CRDTs and compared them theoretically. Our contribution to this field of research is a comparison of OT and CRDTs implemented in Overleaf, a document editing software product. The research that is discussed in this paper was built off of the current published version of Overleaf. Since the current implementation utilizes the operational transform approach to real-time collaborative document editing, it allowed a baseline of an OT

implementation to be evaluated and compared to a different version of implementing real-time collaboration through CRDTs.

Overleaf's public source code is broken down into multiple docker containers. The docker containers allow for Overleaf to separate various parts of the product as a whole into nested containers of all the sections' code and dependencies, thus allowing for easier development upon each section. Overleaf was created by hundreds of engineers, thus it is a large repository of code. Even each individual docker container is large and takes time to understand. Upon research and evaluation of the various docker containers which belong to Overleaf, we discovered that the code which enables collaboration amongst clients working on the same document in Overleaf lies within the `document-updater` container. In document-updater, updates to the document are received, processed and propagated to all clients individual copies of the document. Overleaf performs update handling with an operational transform library called ShareJs. ShareJs is how Overleaf ensures consistency amongst all client copies of the document and the client updates. Since Overleaf already maintains a real-time collaborative editing functionality using ShareJS for an operational transform algorithm, we did not have to implement the OT algorithm ourselves. Thus, we created our own CRDT implementation to allow distributed real-time collaboration in Overleaf and compared it to the original OT version in Overleaf. The specific details of how we implemented the CRDT algorithm for collaboration in Overleaf are outlined below.

4.2 Conflict-free Replicated Data Type

Previous works which focused on comparison of OT and CRDTs assessed the theoretical performances of the two techniques through various ways including theoretical

timing analysis [28, 27, 25] and theoretical application of OT and CRDT algorithms to existing editing logs and collaboration traces [2]. This thesis contributes a non-theoretical implementation of CRDT handling for collaborative edits amongst a distributed document. In order to change the implementation of update handling in Overleaf, extensive research took place in order to begin to understand the inner workings of Overleaf. As mentioned earlier, Overleaf is a complex piece of software created and maintained by hundreds of engineers, therefore it was a challenge to become familiar with the code base and how it could be manipulated to operate using CRDTs for editing. Once we were able to gain an understanding of how Overleaf operated on a high level, we began to design a plan for implementing CRDTs to handle the collaborative edits on their distributive system.

Overleaf is broken down into multiple docker containers. Each docker container handles a different aspect of the Overleaf application as a whole. The docker container which we focused on was document-updater. This container is used for handling updates on the document by all clients. We originally planned to rewrite this entire container in order to allow for a fully decentralized CRDT collaborative implementation. Upon research into the Overleaf code base, the changes required to rewrite the entire document-updater section were too extensive to be performed by one person. Therefore, we made the decision to leave the current client and server handling in place, but rewrite the algorithm which handled the users updates.

In order to change Overleaf's document update handling from an operational transform algorithm into a conflict-free replicated data structure form, I utilized an already developed CRDT library called Automerge. Automerge is a CRDT data structure library for building collaborative applications in JavaScript [14]. This CRDT data structure library was created by Martin Kleppmann, a Senior Research Associate and Affiliated Lecturer at the University of Cambridge Department of Computer Science

and Technology, along with his colleagues and other fellow contributors. According to their research and publications, Automerge is a working CRDT data structure library which is still undergoing development to continue to improve and expand. The published version of Automerge which we utilize in this thesis allows for a working implementation of the CRDT data structure algorithm in performing updates to a distributed collaborative document. We implemented Automerge's CRDT data structure into the distributed collaboration update handling in Overleaf. Overleaf, as mentioned in the previous section, performs all document update handling using ShareJs, an operational transform library. The usage of a publicly sourced real-time collaboration implementation allows one to create a collaborative platform, without having to worry about the implementation details and correctness of the collaborative algorithm itself. However, the usage of the collaborative library is more than simply importing the library source.

The transition from the ShareJs operational transform implementation to the Automerge conflict-free replicated data type version, was not trivial. It required developing an understanding of how Overleaf's source code operated in its current state. The transition to utilizing Automerge's CRDT data structure from ShareJS's OT structure began by examining the flow from the update which was submitted, to the update processing, to the propagation of the update to all clients and remote hosts. When a user submits and performs a change to the document, the change is processed into an update object. As shown in Figure 4.1, the update object contains the document id (as it is stored in Redis), the operation (positional argument, operation type, text), the document version number, the meta data and the unique update hash. Once the change has been processed into an update object, the update is then sent to get integrated into the document state.

```
{
  doc: '602854939e5120007ac54f5e',
  op: [ { p: 236, i: 'hello' } ],
  v: 1653,
  meta:
   {
     source: 'P.orQ9g-Ok906Mu2hiAABd',
     user_id: '602854759e5120007ac54f59'
   },
  hash: 'aaffd194e09d638dcd4148443e74b32994c1829c'
}
```

Figure 4.1: Example of the JSON update object produced from an incoming change.

The update must first be examined to ensure that it is not a duplicate of an update which has already been processed. If the update is determined to be new, then it is allowed to continue for processing and can be applied to the document model. A model of the document is created upon the first edit, and is maintained and manipulated as new updates are received and processed. Our CRDT implementation does not waste time by creating a new model of the document for each update. This is an improvement over Overleaf's operational transform implementation which creates a new ShareJS model for the document each time an operation is submitted. Therefore, our CRDT implementation saves time and memory by utilizing the same model to represent the document during each update.

A user's text editing operations are broken down into simple insertions or deletions. Thus, no matter what action the user is performing, it can either be defined as an insertion or a deletion. The update maintains the information on the operation type and the text involved in said operation, therefore, the update is then parsed and information is extracted from it in order to perform the update to the document itself. The CRDT algorithm takes the positional operation and converts it into an identifier operation to be added to its internal identifier list. If the operation is an insertion,

the identifier is added. If it is a deletion, then the identifier remains in the internal list, but it becomes a tombstone. Deleted operations remain in the internal list in order to allow for updates to maintain consistency. Once the operation is applied and added to the document contents, there are essential checks and recordings which must be performed in order to ensure consistency. To ensure updates are performed casually, the current time of operation execution must be recorded and incorporated into the update meta data to be returned. Next a new hash must be computed and compared to the original update objects hash for guarantee that the file was not corrupted during the update. Finally, the version of the document needs to be updated according to the number of operations performed during the update. Once all steps have been performed and the update object has received all necessary new data, it can be returned and propagated to all remote clients. This system in theory sounds simple, but ensuring consistency amongst all remote clients is far from easy.

The Automerge library is supposed to ensure correctness, but the usage of the library requires many nuances to be implemented correctly. There are some faults in the current CRDT implementation when integrating the new algorithm into Overleaf. In order to test our implementation to see how well the CRDT algorithm could handle multiple users performing edits on the same document, we developed multiple scenarios to see what would happen. These scenarios are detailed in the next chapter. During our testing we found that most of the time the edits were executed and propagated to all clients smoothly. However, sometimes a notification appeared that the documents had gone out of sync and a refresh needed to take place. This did not happen often, but we wanted to find out why it was happening. We narrowed down the refresh alert occurrence to times in which operations were submitted at the same time. Details on this testing and possible reasons for this special case are discussed in the next chapter.

When examining our implementation as a whole, we found that semi-realistic collaboration can be successfully performed. However, as mentioned earlier, if there are operations which are submitted at the exact same time, the current implementation is not able to properly handle both operations. The first operation which is handled by the algorithm is successfully applied and propagated to all clients in close to real-time. However, the second operation which is handled by the algorithm is only applied locally to the user who submitted the update in real-time. The other clients and users receive the update once a page refresh takes place. Once the page automatically refreshes itself, or the user manually refreshes the page, all changes are applied and each of the consistency requirements are met. This fault will be discussed in further detail in the next section.

In order for changes to be applied and propagated to all clients, the current Overleaf document system requires locks to be in place. There is a central database which holds the master copies of the document. When a change is going to be applied, a lock for the respective document must be acquired. Once the lock is in place, the update may be applied, the lock can be released, and the document change can be propagated to all clients. This centralized system of data is not necessary for a CRDT implementation, but it is how the current system is operating. The possible advancements for this system will be discussed in Chapter 8.

Chapter 5

EVALUATION

The following chapter performs an analysis on the results of OT and CRDT implementations in Overleaf in order to compare their real-time collaborative performance.

5.1 Overview

In this section, we discuss the performance results when testing Overleafs real-time collaboration performance while utilizing an operational transform implementation as well as a conflict-free replicated data type implementation. When evaluating real-time collaboration performance, it is essential to observe the latency of updates as well as the consistency of updates. Consistency in this context has multiple requirements in order to achieve true consistency [24]. The first requirement is causality-preservation. Causality preservation means that the operations which are performed must be executed according to their their causal effect order[25]. Another requirement is version consistency, also known as, convergence. Version consistency is achieved when all versions and remote replicas have the same resulting effects of the same set of performed operations, meaning that all changes performed on the document replicas affect all document replicas the same way and produce the same resulting document. The final requirement for consistency is intention-preservation. Intention preservation is achieved when the updates that are performed on the versions of the document result in the updates which the user intended to be performed. This could be messed up if user A intends to write "dog" at position 3 of a document, while user B intends to simultaneously write "cat" in position 3 of a document. The users would expect

their operations to result in the document reading "dog cat" or "cat dog" or perhaps if there was a high enough latency each individual user would only see their own respective changes. However, if the operations became jumbled, the resulting edit on all versions of the document may differ from the users' intended result. For example, the simultaneous operations could result in "docagt". Despite both users seeing the same consistent result, their intention preservation does not prevail. It is essential that real-time collaboration algorithms are capable of maintaining intention preservation in order to allow for productive and useful collaboration. If each of the aforementioned requirements for consistency are met, then the real-time collaboration algorithm can be considered to maintain consistency. In order to effectively evaluate real-time collaboration using the OT and CRDT algorithms, the performance evaluation focused on each algorithms update latency, as well as how well each algorithm met the requirements for consistency.

5.2 Testing Plan

The testing of OT and CRDT performance in Overleaf was executed on a Intel® Core™ i7-6600U CPU @2.60GHz, 2808 Mhz processor with two cores and four logical processors. The measurements recorded include latency for single character insertion, latency of single character deletion, as well as update consistency amongst local user copies of the document. The free open source version of Overleaf which we are working with does not allow end to end testing such as a simulation of multiple threads running and manipulating the same document. Due to this limitation we had to get creative in our validation of the basic insertion and deletion functionality as well as how we examined the consistency results from user edits. We decided that the closest and most reliable way to examine the consistency is through unit tests as well as visual observations of the real-time collaboration between two users. By visual

observations of real-time collaboration between two users, we mean that we examine the resulting documents for both users after edits are performed. This allows us to examine if the operations result in the intended effects, if the documents all converge to the same result and if the edits are being applied in the correct way. In order to thoroughly examine the consistency, there will be multiple examined scenarios. Scenario 1 will involve two users simultaneously editing separate sections of the document, thus producing no conflicts. Scenario 2 will involve two users simultaneously editing the document in the same position in order to intentionally cause conflicts. Scenario 3 will involve users simultaneously adding items to a list in the same position, thus causing a conflict. Each of the aforementioned scenarios will take place in a round of insertion operations as well as a round of deletion operations. The final scenario, scenario 4 will involve a more realistic collaboration scenario where users are tasked with typing up a piece of text together and do not maintain any other form of communication with each other besides the mouse location indicator for all collaborating users on the document.

5.3 Performance of Operational Transform

The operational transform algorithm has been around for much longer than the conflict-free replicated data type algorithm has. Operational transform has long been regarded as the industry standard for implementing real-time collaboration thus resulting in most, if not all, major commercial collaborative products utilizing operational transform in their implementations. Overleaf's published version had operational transform implemented as its algorithm for allowing real-time collaboration. Due to the fact that Overleaf and many other large collaborative products utilize operational transform, we predicted that there would be high consistency results when testing the consistency of the operational transform algorithm.

In order to gain a baseline of how operational transform performs, we focused on user operations in the form of character insertions and character deletions. All real-time collaboration on documents can essentially be broken down into insertions and deletions. Overleaf has a source code section where users can collaboratively edit the source code together which can be compiled into the resulting document. In order to test the latency of updates between collaborative users, we timed the background latency of how long it takes for the updates each user makes to be registered on the other users versions. This was first measured with only one user performing updates at a time, then it was later tested with both users simultaneously making updates to the document. When performing latency tests on one user updating the document, we measured the average time to update for insertion operations as well as deletion operations. Based on an average of 50 insertions from a single user in a document, a single user insertion of a single character has an average latency of 2.941 ms. Based on the average of 50 deletions from a single user in a document, a single user deletion of a single character has an average latency of 3.146 ms. In order to observe if the latency timings were affected by multiple users editing the document at once, we performed the same latency tests while two users were working together to perform the operations. When two users were working to produce 50 insertions to the document, there was an average insertion latency of a single character of 2.869 ms. When two users were working together to produce 50 deletions in the document, there was an average deletion latency of a single character of 3.188 ms. The aforementioned testing scenarios which were performed demonstrate the latency timings for the operational transform algorithm under the conditions which our computer was facing at that time. Therefore these timings may vary depending on the internet connection at the time of testing, as well as what processes are being ran on the computer at the same time. These timings do give us insight into a potential average latency timing for the OT algorithm under the various scenarios. Figure 5.1 gives a visual comparison

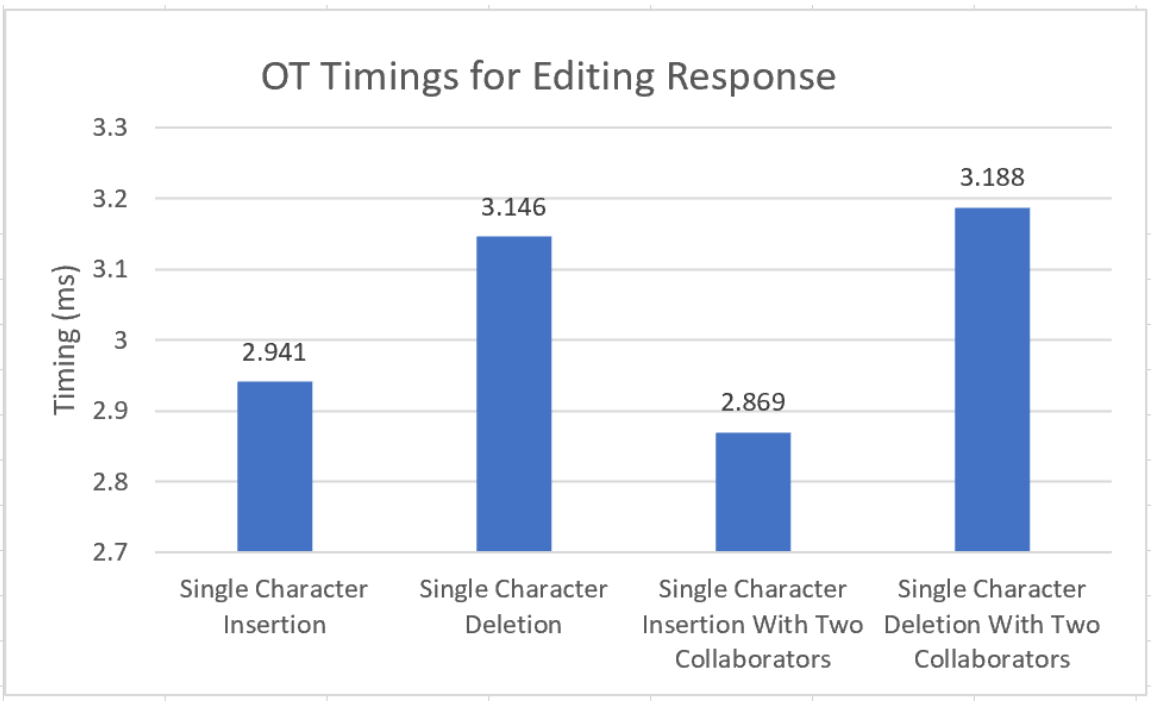


Figure 5.1: This figure provides a visual representation of the timing results for each of the editing scenarios utilized when examining the performance of the operational transform algorithm.

of the timing results for each of the scenarios tested using the operational transform algorithm which was originally implemented in Overleaf.

Consistency of the operational transform algorithm for real-time collaboration was also tested through visual examinations of two user's local editing environments. During this testing we also broke down most of the consistency scenarios into insertions and deletions, thus, each of the discussed experiments took place in an insertion version as well as a deletion version. We first had the users perform insertions and deletions in two different locations of the document. This experiment met each of the three requirements for consistency: causality preservation, version consistency and intention preservation. When the users were performing insertions and deletions at different positions of the document, all operations executed the expected effect, and all replicated copies remained consistent. Next we tested the consistency of the algorithm when purposely creating conflicts by performing insertion and deletion op-

erations in the same position by both users. This also resulted in full consistency. We found no errors in intention or consistency when performing operations in the same position at the same time when testing the operational transform algorithm. Next we wanted to examine the effects on consistency when working with lists using the operational transform algorithm. In order to try and cause conflicts in the list ordering, the users simultaneously (at a human scale) attempted to perform the necessary insertion and deletion operation in the same position of the list. When working with lists, the operational transform algorithm continued to demonstrate consistency. During the specific scenarios which we tested Overleaf's original operational transform implementation for handling real-time collaborative updates, the results were as expected, and the algorithm performed well and met all consistency requirements.

As a final evaluation of operational transform's performance, we tested the consistency in a more realistic collaborative editing scenario. We tasked the two users in typing up a defined text in the document. The users were unable to discuss or collaborate outside of their editing environments. This scenario was intended to simulate a more realistic editing scenario in which users are performing insertion and deletion operations as they naturally would when collaborating on a text document, with no boundaries for potential conflicting user operations. The resulting updates and documents at the end of the testing scenario were as intended, and the documents were consistent with all copies. Operational transform performed well in this experiment and did achieve all requirements for consistency.

5.4 Performance of Conflict-free Replicated Data Types

Conflict-free replicated data types are the more recent idea behind implementing real-time collaboration in comparison to when operational transform was founded.

Since CRDTs were first proposed, there have been claims of the superiority of CRDT implementations over OT implementations, however, much of the previous research which has been performed has not been able to prove or demonstrate the superiority of CRDT over OT. The work which has been performed before has been theoretical analysis comparing OT and CRDT algorithms. The purpose of implementing a CRDT to perform the collaborative edit updates to a document was to allow for a real implementation of a CRDT in a published product which was originally running using an OT algorithm. At first thought we expected CRDTs to have better consistency and update performance, however, after performing research and studying previous works, our expectations for the consistency and timing performance of CRDTs were lowered. In order to evaluate and understand our CRDT implementation in Overleaf, we followed our testing plan and evaluated the performance of the CRDT algorithm in a variety of situations which challenge the CRDT to expose implementation and algorithmic flaws if any exist.

Our CRDT implementation which handled collaborative edits in Overleaf holistically did not perform as well as the original OT algorithm. The exact results of both the timing as well as the consistency of the CRDT algorithm in specific editing and collaboration scenarios will be discussed in this section. Possible reasons and explanations for the performance will be discussed in the next chapter.

We first began our evaluation of the CRDT performance by examining the update latency just as we did for our evaluation of OT performance. The update latency in this context which we examined was the timing for the users update to be processed and incorporated into all users replicated document editing sites. We performed the CRDT and OT latency timing testing immediately after each other in order to try and emulate the same testing environment for each. As mentioned earlier, latency can be affected by many factors such as internet connection and background processes

on a computer. Therefore these timings more so provide insight into how the CRDT and OT latency timings compare under similar environments. We followed the same testing scenarios to evaluate the CRDT performance as we did to evaluate the OT performance. The first scenarios were simple single character insertions and deletions to the document by a single user. The latency for a single character insertion using the CRDT algorithm based on the average timing for 50 individual character insertions was 3.390 ms. The latency for a single character deletion using the CRDT algorithm based on the average timing for 50 individual character deletions was 3.415 ms. The next scenario which we used to evaluate algorithmic update latency was two users performing insertions and deletions at the same time. We tested this scenario in order to observe if there was a change in timing latency when more users were performing edits to the document. The latency for a single character insertion based on an average of 50 insertions performed by two users was 3.009 ms. The latency for a single character deletion based on an average of 50 deletions performed by two users was 3.209 ms. A visual representation of these timings is reflected in Figure 5.1. The CRDT latency timings were similar to the OT latency timings in the same given scenarios. Further analysis and discussion of the latency results will be discussed in the next chapter.

After evaluating the latency performance of our CRDT implementation for handling collaborative edits, we evaluated the CRDT implementations consistency. We used the same scenarios to test the consistency as were mentioned in the testing plan section. During our evaluation of the CRDT implementations consistency, we found that the CRDT algorithm was able to reach eventual consistency in most cases. When inserting into separate positions in the document with no conflicts, the CRDT algorithm met all consistency requirements. When operations were not performed simultaneously, the updates were executed and appeared in close to real-time and met all consistency requirements. However, in each of the aforementioned scenarios, the

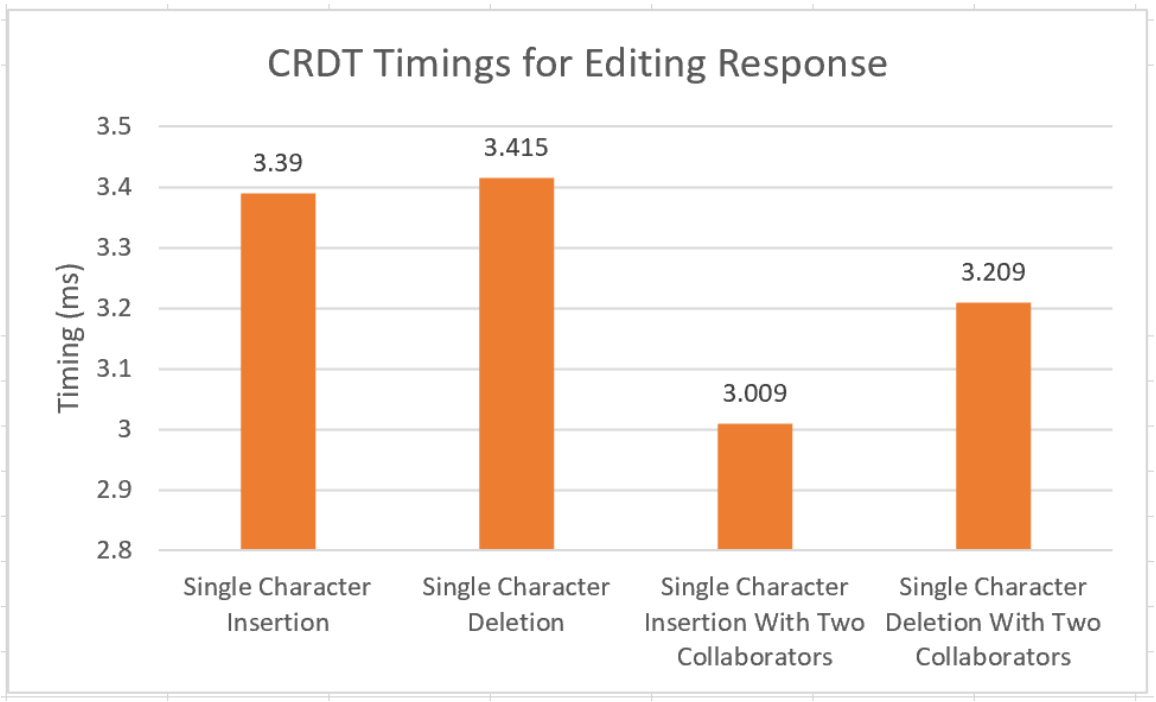


Figure 5.2: This figure provides a visual representation of the timing results for each of the editing scenarios utilized when examining the performance of the conflict-free replicated data type algorithm.

CRDT algorithm met eventual consistency. This meant that when users performed concurrent operations, the operations were applied locally in real-time, but the edits were propagated and updated on the replicated copies once a refresh took place. Once the refresh took place all requirements for consistency were met. When two users inserted into the same location or deleted from the same location simultaneously, there was eventual consistency. The same held true for concurrent insertions and deletions for the same location in a list. The requirement for a refresh when concurrent operations take place does not allow for smooth collaboration amongst remote clients. The impact and possible reasons for this eventual consistency will be discussed in the next chapter.

The final scenario which was tested for the CRDT algorithm was a realistic collaboration scenario where two users were tasked with typing up a piece of text into the document. During this scenario the users were able observe each others edits in

real-time and successfully perform the collaboration. If the users performed exactly concurrent edits, the editing environment notifies them that the documents are out of sync and a refresh is required. Once the refresh took place, the updates were all applied and consistency was maintained. This allowed for collaboration to take place amongst the clients. Therefore, the CRDT algorithmic implementation would need to continue to be developed in order for it to compete with the original OT algorithm for effectiveness.

Chapter 6

DISCUSSION

This chapter discusses the results of testing the real-time collaboration performance in Overleaf using OT and CRDT algorithms.

6.1 OT and CRDT Claims

When researching operational transform and conflict-free replicated data types, there are the same general claims and statements which continue to appear. In most articles OT is referred to as the standard practice for implementing real-time collaboration, and CRDT is referred to as the newer way of implementing real-time collaboration which has not been revealed to be implemented in any major collaborative products. Despite the lack of CRDT implementations, there are many claims of superiority of CRDT over OT [24, 25, 27, 28]. Some researchers have sought to verify these claims [2, 25, 27, 28]. The previous research has all been theoretical, or a simulation of how the algorithms would execute and perform on existing editing logs. Our research sought to implement a CRDT algorithm in Overleaf and evaluate its performance. Through this implementation and evaluation we hoped to compare the theoretical findings to the real implementation results, as well as discuss the observed advantages and disadvantages of OT and CRDTs. The following sections will discuss each of the aforementioned goals and their results.

6.2 Comparison of the OT and CRDT Implementations

Overleaf is a widely used product which has been developed for many years now by many engineers. It implements its real-time collaboration through operational transform, the industry standard for implementing distributed collaboration. When first beginning to work with Overleaf and examining its published product, the collaborative aspect of its editing functionality appeared to work well for a typical collaborative setting between two people, therefore we assumed the OT algorithm would be well-performing. We also acknowledged that Overleaf as a whole was a very large and complicated repository of code. This was a major hurdle which had to be tackled in order to begin any work on this project. When reaching out to the engineers who created Overleaf, their responses were skeptical in the possibility for one person to implement a CRDT algorithm in Overleaf to replace the OT algorithm. Their skepticism motivated this thesis to continue researching into the topic of CRDTs and to continue working with Overleaf source code to see what we could accomplish. There were many times when the project seemed impossible, yet we continued to persevere. The resulting implementation focused on the handling and application of updates to the existing document structure through a CRDT. Final results and evaluation demonstrated that the CRDT implementation was fairly well performing and could be further developed to be just as well performing as OT. There is still room for development and research into CRDT. At this time using OT for implementing distributed collaboration is the safer and more reliable choice.

In our testing and evaluation of the OT implementation which was already a part of Overleaf, our assumptions were proved true. The first testing mechanism for OT was through unit tests. The OT implementation passed all unit tests to ensure basic insertion and deletion functionality correctness. We also included observational

testing using the web interface of Overleaf to see the correctness of the results given various editing situations. As explained in the evaluation chapter, Overleaf's original OT implementation performed well and in each editing scenario it maintained all three consistency requirements: convergence, intention preservation and causality preservation. Since Overleaf is a published product with a pro version that can be paid for, we expected it to be well performing with few noticeable errors to the average user. Since we were beginning with a baseline implementation which already performed well, it was a challenge to create an implementation to compete with the original. Our CRDT implementation was able to reach similar results to that of the OT implementation, with the exception of a few special cases which will be discussed.

Table 6.1 displays the results of our visual consistency checks amongst various editing scenarios for the OT implementation as well as the CRDT implementation. We created multiple editing scenarios to demonstrate if consistency was maintained. From our testing we found that the operational transform algorithm was able to maintain consistency in all scenarios including times when users performed concurrent operations in both the same and different locations of the document. This demonstrated that the operational transform algorithm was very well performing and is able to soundly handle conflicts without the clients noticing there were any. When examining the CRDT implementation, we found that the CRDT algorithm was able to maintain consistency and allow for a realistic editing scenario in most cases. When two users were editing the same document in their respective replicated editing sites, their edits were executed and propagated to the other clients smoothly as long as the edits were not performed at the exact same time. The special case of completely concurrent edits caused the clients to be notified that the files had gone out of sync and a refresh was needed. Once the refresh took place, both edits were preserved and included in the document and the files maintained consistency. This is an important observation because despite the need for a refresh to take place in order to

handle concurrent operations, the edits of both users were maintained and not lost, and the documents remained consistent. Despite our CRDT implementations ability to maintain consistency after a refresh, there is still concern over how CRDTs handle a server failure or an unreliable internet connection. In the future work chapter we mention this as an area where further development could take place to enhance the implementation and allow for more smooth collaboration when using the CRDT implementation.

There are a few possible reasons as to why the CRDT implementation is not able to handle the special case of two exactly concurrent updates taking place. One of these possible reasons is that there may be a missing event alert. It is possible that the updates each take place locally at the same time, but since each local document version was busy performing their own update, the other editing clients are not made aware of the other editing clients updates until a refresh takes place. Therefore an event handler may be waiting until the local updates take place and a refresh happens before the event handler is able to ensure all updates have been propagated to all clients. This could possibly be solved by creating an update queue. There would have to be an algorithm in place which handles the decision of which ordering the concurrent updates are placed in the queue. The algorithm would have to produce the same results each time when given the same inputs in order for this to work. Or else the algorithm may produce different results on different editing clients and thus result in inconsistency. With the editing queue, I imagine that an update would be able to take place, be propagated to the clients and then the algorithm can handle the next update in the queue. This idea may allow for all updates to be handled and propagated to the others without the need for a refresh. Due to time constraints, this thesis investigated this possible fix, but was not able to successfully implement it as a solution.

Table 6.1: This table demonstrates the performance of the OT algorithm as well as the CRDT algorithm under various scenarios when evaluated on their ability to meet all requirements for true consistency.

Scenario	Intention Preservation (OT)	Convergence (OT)	Intention Preservation (CRDT)	Convergence (CRDT)
Inserting in separate positions with no conflicts	Yes	Yes	Yes	Yes
Inserting in the same location with conflicts	Yes	Yes	Yes *	Yes *
Deleting in separate locations with no conflicts	Yes	Yes	Yes	Yes
Deleting in the same location with conflicts	Yes	Yes	Yes *	Yes *
Inserting into a list at the same time with conflicts	Yes	Yes	Yes *	Yes *
Deleting from a list at the same time with conflicts	Yes	Yes	Yes *	Yes *
Realistic collaboration scenario with no collaboration outside of the document	Yes	Yes	Yes **	Yes **

* Eventual consistency is reached on all replicas after a refresh takes place

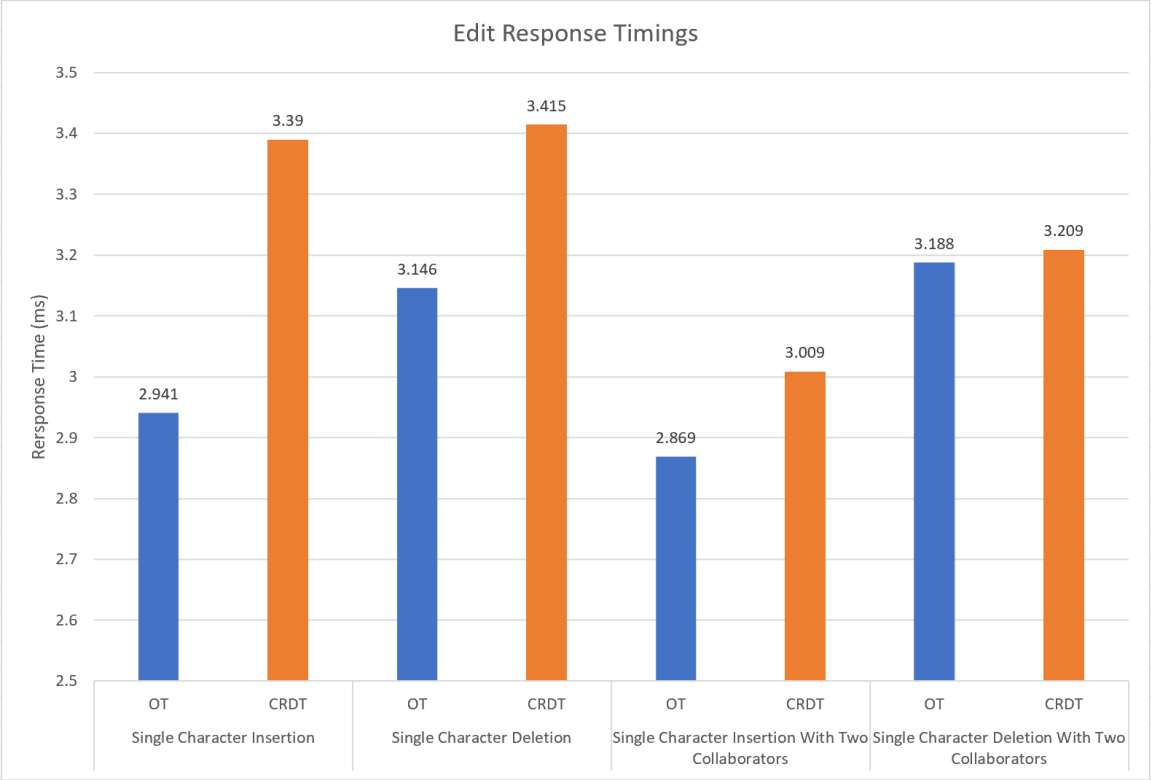
** Consistency is maintained amongst updates in near to real time unless the users perform Concurrent edits. In that case the edits perform eventual consistency after a refresh.

Further comparison between the OT and CRDT implementations can be focused on their respective latency timing for edits. Table 6.2 displays the edit latency's for the OT and CRDT algorithms in various editing scenarios. For the context of this testing and evaluation of edit latency timings, we define the edit latency as the time it takes from the edit first being received and processed till the time that the update has been applied to the master document contained on the central server. The first editing scenarios which were examined were single character insertions and deletions

performed by one user. For the OT implementation the average insertion latency for one character was 2.941 ms and the average deletion latency for one character was 3.146 ms. In comparison, the CRDT implementation had an average insertion latency for a single character of 3.390 ms and an average deletion latency for a single character of 3.415 ms. We also tested the edit latency timings when there were two collaborators contributing the insertions and deletions to see if the added client affected the timings. When there were two collaborators performing the insertions and deletions the operational transform algorithm had an average single character insertion latency of 2.869 ms and an average single character deletion latency of 3.188 ms. When there were two collaborators performing the insertions and deletions for the conflict-free replicated data type algorithm, the average single character insertion latency was 3.009 ms and the average single character deletion latency was 3.209 ms.

The edit latency timings for the OT implementation and the CRDT implementation were very similar in all scenarios. We found that in each scenario the CRDT implementation had a slower edit latency than the OT implementation. The theoretical research which has been done came to the conclusion that the CRDT algorithm is theoretically slower than the OT algorithm [28, 27, 25]. Part of the reasoning for the CRDT algorithm being slower is that the algorithm timing is affected by the starting state of the document before an edit is performed. The CRDT algorithm must populate its model with the contents of the original document before an initial edit can be performed. This is not the same for the operational transform algorithm. For OT the original contents do not matter. Thus, the original document could be thousands of lines long and it wont affect the OT timing, but it will affect the CRDT timing. Another reason is that the CRDT algorithm performs its edits by counting and traversing the number of live objects, the search-count method. This alone is a O(n) worst case scenario, but there are not only live objects to traverse and check, but also tombstone objects which have been deleted from the document. These tombstone

Table 6.2: This table provides a clear comparison of the timing performance of the edit latency's for the OT and CRDT algorithms under various editing scenarios.



objects could theoretically be removed through garbage collection, however, this is hard to implement in practice and is yet to be included in the Automerge CRDT implementation [13]. The CRDT algorithm is all identifier-based, each of the data objects contains the respective unique identifier of that object as well as the respective unique identifiers of its neighboring objects whether they are live or dead. Therefore if tombstones were removed, neighboring data objects would be corrupted and their now missing neighboring identifiers would need to be updated as well.

6.3 Advantages and Disadvantages of OT and CRDTs

This section will discuss the advantages and disadvantages of implementing real-time collaborative editing on a text document using when respectively using an OT implementation and a CRDT implementation.

6.3.1 OT

Operational transform has been the traditional replication mechanism for concurrent document editing for many years now [2, 11, 16]. Due to its long standing reputation as the traditional standard for concurrent document editing, there have been many products which use OT in its collaborative platform such as Google Docs, Google Colab and Jupyter [13]. Having highly regarded products utilize OT as its replication mechanism gives OT the advantage of being respected and having many people who have investigated the algorithm and worked on solutions for many underlying problems it may have in order to ensure easy collaboration of OT products. Along with a more established reputation than CRDTs, OT algorithms also have faster latency timings. Theoretical research has demonstrated that the edit latency for OT algorithms are faster than those of CRDT algorithms [27]. This point of faster latency timings was also demonstrated in the work of this thesis when implementing a CRDT handling of text document edits to compare to the OT handling of text document edits. The timings which came from the experiment demonstrated that the OT algorithm appears to have a shorter latency time.

Some disadvantages of OT as the replication mechanism for concurrent document editing include its centralization. OT relies on a centralized server for its network communication. This does not allow for a peer to peer network and also means that

there could be personal information stored on a large central server. Large corporations which utilize centralized approaches to store the shared documents, may also be storing personal information about the documents users as well. This could be considered a privacy threat to users of these corporations [2]. There have been studies which claim that OT does not require a centralized server [25, 27, 28], but to our knowledge a non-centralized OT replication mechanism is yet to be implemented in a real product. Another disadvantage due to the transformation requirement amongst concurrent updates in OT is that OT is not very scalable, meaning OT does not perform well when there are a large number of users performing concurrent operations [2]. This lack of scalability for large numbers of collaborators is typically not of concern because the number of users simultaneously collaborating on a single document does not typically reach OTs limitations. There is a certain point in which there are simply too many collaborators on a single document for the collaboration to still be effective.

6.3.2 CRDT

Since CRDTs are the more recently proposed replication mechanism for concurrent document editing, there is not as much trust in their ability nor as many examples of products which use CRDTs. Even with CRDTs being proposed more recently than OT, there is one advantage of CRDTs which commonly appears in literature: CRDTs do not require a centralized server for network communication. The CRDT library, Automerge, which was utilized in this thesis, is an example of a CDRT algorithm which does not require a central server. The only network requirement for Automerge is that all messages which are sent by a replica are eventually received by all other replicas [13]. This essentially means that when a CRDT is used as the replication mechanism in a text editor, a user can perform edits offline which will be

successfully merged with all other replicas of the document once back online while still maintaining consistency. CRDTs rely on the mathematical properties of commutativity, associativity and idempotency to guarantee that all replicas of any CRDT will be able to converge in a consistent manner despite failures [22]. Much of the research which has been performed prior believes that using a CRDT for distributed real-time collaboration could be very successful. The work performed in this thesis demonstrates that even when using CRDTs to handle the edit processing, the CRDT implementation performs similarly to the OT implementation. Their timing differences are minuscule when observing with the human eye during a collaborative session. Previous research has claimed that people comfortably work with changes in collaborative settings when the response latency is less than 50ms [23, 12]. This claim demonstrates that both the CRDT and the OT implementations performed well within the limit for avoiding frustration amongst users during a collaborative session. We believe that if the CRDT mechanism and ideology of server decentralization was fully implemented, the CRDT implementation would out perform the OT implementation. Due to time constraints this could not be proven in this thesis, but this is an area of development which is discussed in the future work chapter.

Despite advantages of CRDTs, there are still some disadvantages which must be understood and considered. CRDTs for text editing rely on an ordered list of characters to represent the text document. Each time a character is added to the text document, it is also added to the ordered list. However, when a character is removed or deleted from the text document, it remains in the list but is marked as invisible and stays as a tombstone in the ordered list. This means that the CRDT data structure behind the collaboration is exponentially growing and could produce slower and slower times for very large documents. A possible solution for this would be to implement a garbage collection algorithm to get rid of tombstones. Automerge does not currently have garbage collection, but Dr. Kleppmann states that it is a work in progress which

the Automerge team hopes to release at some point [13]. The difficulty of garbage collection is the inherent data structures which lie inside each object in the CRDTs ordered list. Each character object in the list is identified by [(character), (id), (idNeighborBefore), (idNeighborAfter), (visibility)]. If a tombstone were removed, but it has live, visible neighbors, then the neighbors internal object structure would have an invalid id for its neighbors. This would cause issues in correctness of the algorithm. A possible solution to this problem is to only remove tombstones which are also neighbored by tombstones on both sides. If this were the case we believe that the incorrect neighbors for the remaining tombstones would not cause issues since they are not live in the document. Another possible solution could be to update the neighboring id's when removing a tombstone. Both of the aforementioned solutions are merely ideas and have not been proven or attempted in this thesis.

6.4 Technical Problems In Distributed Collaboration Tools

When dealing with any type of large distributed system, replication and consistency are essential features of that system [22]. The CAP theorem states that it is impossible to simultaneously ensure strong consistency (c), availability (a) and tolerate network partition (p) [22]. Strong consistency is when all distributed server nodes always contain the same, most up to date data. In order for this to be possible, when one server node or client is editing the system or piece of data, there must be a lock in place which does not allow any other nodes to contribute or make changes. This locking requirement does not allow for availability, thus demonstrating the CAP theorem. Therefore, distributed collaborative systems must rely on the principle of eventual consistency, in which all server nodes will eventually be caught up and contain consistent data. As mentioned before, in order to allow for efficient and effective collaboration, this eventual consistency must happen very quickly. Performance is

a key factor when building distributed collaborative tools [13, 2, 6]. If updates and changes are taking too long to be processed and propagated, the users will become frustrated and will not have a good experience.

Due to the aforementioned requirements of collaborative tools, When creating/building distributed real-time collaboration there are a few problems which always come up and need to be dealt with. One of the problems which come up are concurrent edits which take place at the exact same time by two different clients on their individual server nodes. One possible easy solution to dealing with conflicts would be to discard all concurrent modifications. The problem with this solution is data loss [13]. It is important that the intended modification which a user executes is able to be performed on the document. If users were continuously unable to have their changes execute, they would not want to continue working with the editing platform.

The way operational transform algorithms typically handle this problem is through operational transformations. The back end algorithm first locally applies the change, then in its transformation stage it examines how the concurrent operations affect each other and then adjusts the positional-based operations as needed. Once the operational transformations take place, the operations are then propagated to all other clients.

CRDT algorithm handles concurrent operations by first applying the update locally, just as operational transform does. Then, the back-end processing applies the concurrent change by examining the positional argument and counting the number of objects in the CRDTs ordered list before any of the concurrent operations took place. This allows the underlying algorithm to determine the operations affect without the other operations altering its result. If two concurrent operations affect each other, they are adjusted accordingly and able to be merged. If inserting into the same position, there are built in algorithmic decisions which determine which operation goes first. This

ensures all data is not lost, and that no matter what is determining the operational ordering, the same inputs to the decision algorithm produce the same results each time.

Another problem which appears with distributed collaborative editing when specifically working with CRDTs, is the need for unique identifiers. Each CRDT implementation must develop a system of creating reliable and unique identifiers. Using numerical identifiers has its limits which can be exceeded when working with extremely large documents. The way that the CRDT library Automerge handles this problem in their implementation is through the use of lamport timestamps as their unique identifiers. Lamport time stamps are a pair (c,p) in which p is the replica or client id of the respective client. This id is typically a hash of the public key. The c is a counter which is stored uniquely at each replica and keeps count of the total number of operations performed. Since each replicas number of operations is strictly monotonically increasing, the lamport time stamp is believed to always be unique [13]. Based on our research in this thesis, Automerge's claim to have found a solution to the unique identifier problem seems to be correct. We are unable to identify a problem with their method at this time.

Network connectivity and failure is a problem with all distributed systems. It is essential for all distributed systems to be fault tolerant and successful in maintaining availability in the system even with failure. This is where our current CRDT implementation in Overleaf fails. Our implementation works well except when concurrent operations are performed. In this case, the web page must be reloaded in order to have the concurrent operations applied and for the replicas to remain consistent. This means that during the refresh, the system is unavailable to be worked on. This problem is an area for future work beyond the time constraints of this thesis.

Chapter 7

CONCLUSION

The following chapter summarizes much of the research contained in this thesis, as well as gives final concluding remarks on distributed real-time collaboration.

7.1 Distributed Real-Time Collaboration

Effective and efficient real-time collaboration is now more important than ever. During the COVID-19 pandemic most work places had to switch to remote work seemingly overnight. This meant that teams which were previously working together in person, now had to find ways to effectively collaborate and communicate online. One of the major tools which allows for effective virtual teamwork is distributed real-time collaboration. No matter the tool which people choose to utilize when collaborating online, there must be some sort of distributed collaboration algorithm in place which allows for the application to perform. This thesis focused on researching and understanding two major methods of implementing real-time collaboration through an operational transform algorithm and a Conflict-free Replicated Data Type algorithm. The long respected operational transform algorithm is what has been seen in most if not all major collaborative services. There has been a lot of research which has taken place on OT, as well as some theoretical research on the CRDT algorithm. This thesis set out to create a real implementation of a CRDT algorithm in a collaborative product. The resulting implementation using a CRDT algorithm as the consistency protocol in Overleaf resulted in successful edit handling from multiple users except when the edits were performed at close to concurrent times. This special case of concurrent

operations being performed requires more precise exact timing than we would have expected. During testing there were times where we intended to purposefully create concurrent edits which were registered as non-concurrent. Future work could investigate how often edits to a collaborative document are registered as concurrent edits in a realistic editing scenario. Our latency timing provides insight into how our CRDT and OT algorithms performed under similar environments. These timings could be affected by poor network connection, background processes, etc. Therefore our research provides new insight into CRDT performance, but we conclude that at this time in CRDT development, OT is still the more reliable methodology for implementing distributed real-time collaboration.

The arguably most important goal of distributed real-time collaborative tools is that the shared document should be just as easy to edit as is a single author document [2]. Throughout this thesis there is a lot of discussion surrounding how to make the collaborative environment as smooth and easy to work in as possible. Operational transform has received a large amount of theoretical research as well as real world implementations. Throughout the theoretical and non theoretical research, the operational transform mechanism for implementing real-time collaboration has been well refined. Many problems which come with collaborative platforms such as concurrent edits, data preservation and network availability have been researched and resolved in OT implementations. CRDTs on the other hand have not received nearly the same extent of research and development. Much of the previous research has been largely theoretical with claims of superiority over the OT methodology. However, these claims for the most part are merely claims. As demonstrated by this thesis, CRDTs have similar performance to OT mechanisms when working with small sized documents under similar conditions and handling editing updates using the OT and CRDT algorithms. Due to time constraints this thesis implemented the algorithmic approach of CRDTs to handle document edits, but it did not implement the decentralized

server methodology behind CRDTs. Previous research claims that the decentralization ability of CRDTs makes them superior to OT implementations. This claim still remains to be validated. This thesis gathers many pieces of research on mechanisms for allowing distributed collaboration on text documents, as well as discusses previous findings, the advantages and disadvantages of OT and CRDT algorithms and provides implementation and a discussion of a CRDT algorithm integrated into Overleaf, a public collaborative editing product.

Overall, although OT has been long regarded as the standard for implementing real-time collaboration, this thesis demonstrates that CRDTs should not be written off as a failed idea. With continued research and development it is possible that CRDTs could one day become refined enough to allow for effective distributed collaboration. For now, if one desires to create a distributed collaborative platform, OT is the right decision if looking for a well respected and tested mechanism for collaboration.

Chapter 8

FUTURE WORK

This chapter will discuss possible paths to further the research described in this thesis, as well as possible areas of research to continue to expand upon CRDTs and real-time collaboration algorithms in general. The original goal of this thesis was to create a real, non-theoretical implementation of a CRDT algorithm, as well as research real-time collaborative tools in general. These original goals were met, however, there are still areas and topics which can continue to be researched, as well as expansion and work which can be done to the CRDT implementation of Overleaf. Due to limited time the future work which is outlined here was not able to be completed.

8.1 Expansion of the CRDT Implementation in Overleaf

This thesis presented the results of a CRDT algorithmic approach to handling updates to a shared collaborative document. During the testing and evaluation of the algorithm it was discovered that the CRDT implementation achieved consistency for the most part. The special case in which two users perform exactly concurrent edits failed to be properly propagated to all clients in real-time. This special case required the client to refresh their page in order to see all updates and changes. However, once the page was refreshed the CRDT algorithm demonstrated all requirements for consistency: causality preservation, intention preservation and convergence. Future work could be done to improve the CRDT implementation in Overleaf to not require the pages to refresh in order to process concurrent updates. This would improve the current implementation and allow for smoother collaboration amongst clients. The

closer to real-time in which updates appear on all clients, the better the collaborative environment is.

Another possible expansion to the CRDT implementation in Overleaf is to redo the current back end set up of their server communication. Research would have to be performed which further examines Overleaf and its underlying LaTex environment in order to fully understand if it would be possible to rewrite the back end to not require a centralized server so that collaborative edits can be performed even when the document is offline. From our current research in this thesis I believe that Overleaf would not be suitable in its current state to benefit from a decentralized server. Since Overleaf is an online web service, it appears as though it is a requirement for the document to be online. However, research in this thesis did not focus on this topic, therefore, this is an area which could be looked into in order to expand and perform further research upon this thesis.

Further research could also be performed into allowing garbage collection in CRDT algorithms. Since CRDTs are inherently exponentially growing based on their underlying algorithm, their time and space complexities grow large in relation to the size of the document. Automerge's Dr. Kleppmann states that they are working on and plan to release a garbage collection in the future. Their work, if successful, could be integrated and developed on other CRDT libraries. If the garbage collection is successful, this would address one of the major concerns of CRDT mechanisms and possibly encourage further research and development into CRDTs.

BIBLIOGRAPHY

[1] Cal Poly Github. http://www.github.com/CalPoly.

[2] M. Ahmed-Nacer, C.-L. Ignat, G. Oster, H.-G. Roh, and P. Urso. Evaluating CRDTs for Real-time Document Editing. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 103–112, 2011.

[3] S. B. Bellotti. Walking Away from the Desktop Computer: Distributed Collaboration and Mobility in a Product Design Team. *ACM conference on Computer supported cooperative work*, 1996.

[4] S. Burckhardt. Principles of Eventual Consistency. *Foundations and Trends in Programming Languages*, 1(1-2):1–150, 2014.

[5] S. Cofalik and A. Tomanek. Lessons learned from creating a rich-text editor with real-time collaboration. [Online]. Available from: https://ckeditor.com/blog/Lessons-learned-from-creating-a-rich-text-editor-with-real-time-collaboration/, Oct 2018.

[6] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. *ACM SIGMOD Record*, 18(2):399–407, June 1989.

[7] D. Engelbart and H. Lehtman. Working Together. *Byte*, 13(13):245–252, 1988.

[8] J. Gentle. ShareJS API. [Online]. Available from: https://github.com/josephg/ShareJS. ShareJS.

[9] Google. Colaboratory. Accessed: Jun. 21, 2018. [Online]. Available: https://research.google. com/colaboratory/faq.html. 2018.

[10] J. Hammersley and J. Lees-Miller. Collaborative editing us patent 9,729,672. Google Patents, Aug. 8 2017. US Patent 9,729,672.

[11] P. Hedkvist. Creating a Collaborative Editor. [Online]. Available from: https://www.pierrehedkvist.com/posts/1-creating-a-collaborative-editor, 2019.

[12] C. Jay, M. Glencross, and R. Hubbold. Modeling the Effects of Delayed Haptic and Visual Feedback in a Collaborative Virtual Environment. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 14(2):8–es, 2007.

[13] M. Kleppmann and A. R. Beresford. A Conflict-free Replicated JSON Datatype. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2733–2746, 2017.

[14] M. Kleppmann and A. R. Beresford. Automerge: Realtime Data Sync Between Edge Devices. In *1st UK Mobile, Wearable and Ubiquitous Systems Research Symposium (MobiUK 2018)*, 2018.

[15] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[16] R. Levien. Towards a Unified Theory of Operational Transformation and CRDT. [Online]. Available from: https://medium.com/@raphlinus/towards-a-unified-theory-of-operational-transformation-and-crdt-70485876f72f, 2016.

[17] S. Li, H. Jiang, and M. Shi. Redis-based Web Server Cluster Session Maintaining Technology. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 3065–3069. IEEE, 2017.

[18] P. Nicolaescu, K. Jahns, M. Derntl, and R. Klamma. Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. pages 39–49. Proceedings of the 19th International Conference on Supporting Group Work, 11 2016.

[19] Nunsus. Basic Idea Behind OT. [Online]. Available from: https://commons.wikimedia.org/w/index.php?curid=7123571.

[20] C. Salamanca. The Mother of All Demos. [Online]. Available from: https://escholarship.org/uc/item/91v563kh, 2009.

[21] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-Free Replicated Data Types. In X. Défago, F. Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[22] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[23] B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[24] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-time

Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, 1998.

[25] C. Sun, D. Sun, Agustina, and W. Cai. Real Differences between OT and CRDT under a General Transformation Framework for Consistency Maintenance in Co-Editors. *Proceedings of the ACM on Human-Computer Interaction*, 4:1–26, 2020.

[26] C. Sun, Y. Zhang, X. Jia, and Y. Yang. A Generic Operation Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing systems. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge*, pages 425–434, 1997.

[27] D. Sun, C. Sun, A. Ng, and W. Cai. Real Differences between OT and CRDT in Building Co-Editing Systems and Real World Applications. *arXiv preprint arXiv:1905.01517*, 2019.

[28] D. Sun, C. Sun, A. Ng, and W. Cai. Real Differences between OT and CRDT in Correctness and Complexity for Consistency Maintenance in Co-Editors. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–30, 2020.

[29] G. Tato, M. Bertier, E. Rivière, and C. Tedeschi. ShareLatex on the Edge: Evaluation of the Hybrid Core/Edge Deployment of a Microservices-based Application. In *Proceedings of the 3rd Workshop on Middleware for Edge Clouds & Cloudlets*, pages 8–15, 2018.

[30] R. Yasrab, J. Ferzund, and S. Razzaq. Challenges and issues in collaborative software developments. *arXiv preprint arXiv:1904.00721*, 2019.

APPENDICES

Appendix A

CRDT CODE

```
applyUpdate(project_id, doc_id, update, lines, version, callback){

  if (callback == null)

  {

    callback = function (error, updatedDocLines) {}

  }

  logger.log({ project_id, doc_id, update }, 'applying automerge updates')

  const jobs = []

  const incomingUpdateVersion = update.v

  let dup = false

  //check to see if the update is a duplicate update of one which has been handled
  if(update.dupIfSource)

  {

```

```
      dup = true

    }

    let count = 0

    let appliedOps = {}

    const eventEmitter = new EventEmitter()

    // listener for the event that an operation has been applied
    this._listenForOps(eventEmitter)

    console.time("time to apply")

    //wait till the next tick to ensure all clients receive the
   update
    //waiting lets the system confirm the clients
    return process.nextTick(function() {

      setTimeout(() => 3000)

      //if this is the first operation applied create a new document
    model
      if(!doc)

      {

        console.log("creating a new doc")

        doc = Automerge.init()

```

```
54        doc = Automerge.change(doc, 'Add text', doc => {
55
56          doc.text = new Automerge.Text()
57
58          for (i = 0; i < lines.length; i++)
59
60          {
61
62            let lineLength = lines[i].split('').length
63
64            if(lineLength > 0)
65
66            {
67
68              for(j = 0; j < lineLength; j++)
69
70              {
71
72                doc.text.insertAt(count, lines[i][j])
73
74                count++
75
76              }
77
78            }
79
80            if(i != lines.length - 1)
81
82            {
83
84              doc.text.insertAt(count, '\n')
85
```

```
            count++

          }

        }

        doc.boolValue = false

      })

    }

    //record the old version of the document
    const oldSnapShot = doc.text.toString()

    if (appliedOps[doc_id] == null)

    {

      appliedOps[doc_id] = []

    }

    if (!dup)

    {

      if(update.op)

      {

        appliedOps[doc_id].push(update)
```

```
118
119       }
120
121       for(const op of Array.from(update.op))
122
123       {
124
125         //this is for handling concurrent updates, there needs to
    be position manipulation
126         if (version != update.v)
127
128         {
129
130           if(prevUpdate.d && prevUpdate.p <= op.p)
131
132           {
133
134             op.p -= prevUpdate.d.length
135
136           }
137
138           else if(prevUpdate.i && prevUpdate.p <= op.p)
139
140           {
141
142             op.p += prevUpdate.i.length
143
144           }
145
146         }
147
148         //handling of deletion operations
```

```
149         if(op.d)
150
151         {
152
153           let numDeletes = op.d.length
154
155           doc = Automerge.change(doc, 'deletion op', doc => {
156
157             //wait till the document isnt busy
158             while(doc.boolValue)
159
160             {
161
162             }
163
164             doc.boolValue = true
165
166             for(i = 0; i < numDeletes; i++)
167
168             {
169
170               doc.text.deleteAt(op.p)
171
172             }
173
174             doc.boolValue = false
175
176           })
177
178         }
179
180         //handling of insertion operations
```

```
181         else
182
183         {
184
185           let insertOp = op.i
186
187           let insertArray = insertOp.split('')
188
189           doc = Automerge.change(doc, 'insertion op', doc => {
190
191             //wait till the document isnt busy
192             while(doc.boolValue)
193
194             {
195
196             }
197
198             doc.boolValue = true
199
200             for (j = 0; j < insertArray.length; j++)
201
202             {
203
204               doc.text.insertAt(op.p + j, insertArray[j])
205
206             }
207
208             doc.boolValue = false
209
210           })
211
212         }
```

```
213
214           prevUpdate = op
215
216         }
217
218       }
219
220       else
221
222       {
223
224         //handling duplicate operations
225         metrics.inc('sharejs.already-submitted')
226
227         logger.warn
228
229         (
230
231           { project_id, doc_id, update },
232
233           'op has already been submitted'
234
235         )
236
237         update.dup = true
238
239         eventEmitter.emit('applyOp', project_id, doc_id, update)
240       }
241
242       update.meta.ts = Date.now()
243
244       console.timeEnd("time to apply")
```

```
245
246     //ensuring the document hash is correct and it is not
    corrupted
247     if (update.hash != null && incomingUpdateVersion === version)
248
249     {
250
251       const ourHash = ShareJsUpdateManager._computeHash(doc.text.
    toString())
252
253       if (ourHash !== update.hash)
254
255       {
256
257         metrics.inc('sharejs.hash-fail')
258
259         return callback(new Error('Invalid hash'))
260
261       }
262
263       else
264
265       {
266
267         metrics.inc('sharejs.hash-pass', 0.001)
268
269       }
270
271     }
272
273     const docLines = doc.text.toString().split(/\r\n|\n|\r/)
274
```

```
275     return process.nextTick(function () {
276
277       setTimeout(() => 3000)
278
279       eventEmitter.emit('applyOp', project_id, doc_id, update)
280
281       if (version != update.v)
282       {
283
284         update.v = version
285
286       }
287
288       version += appliedOps[doc_id].length
289
290       eventEmitter.emit('op', update, doc.text.toString(),
    oldSnapShot)
291
292       return callback(
293
294         null,
295
296         docLines,
297
298         version,
299
300         appliedOps[doc_id] || []
301
302       )
303
304     })
305
```

```
306     })
307
308   }
309
310   _listenForOps(eventEmitter) {
311
312     return eventEmitter.on('applyOp', function (project_id, doc_id,
      opData) {
313
314       return ShareJsUpdateManager._sendOp(project_id, doc_id, opData
      )
315
316     })
317
318   },
```